Resolving Constraint Conflicts in the Integration of Entity-Relationship Schemas

Mong Li LEE Tok Wang LING

Department of Information Systems & Computer Science
National University of Singapore
email: {leeml, lingtw}@iscs.nus.sg

Abstract. In this work, we address the problem of constraint conflicts while integrating the conceptual schemas of multiple autonomous databases modeled using the Entity-Relationship (ER) approach. This paper presents a detailed framework to resolve three types of constraint conflicts, domain constraint conflicts, attribute constraint conflicts and relationship constraint conflicts. There are two types of domain constraint conflict, convertible and inconvertible. We distinguish two types of convertible domain constraints conflict, reversible and irreversible, and present an algorithm to resolve domain constraint conflicts. We identify six factors that can contribute to conflict in attribute constraints: imprecise constraint design, domain mismatch, incomplete information, imprecise semantics, value inconsistency and set relation between object types. In relationship constraint conflict resolution, we examine the set relation between equivalent relationship sets and the functional dependencies that hold in these relationship sets. Our conflict resolution approach does not assume that equivalent entity types or relationship sets in two schemas model exactly the same set of instances in the real world. Furthermore, our approach enforces the most precise constraints and enables the retrieval of all the data in the local databases via the integrated schema.

1. Introduction

Schema integration involves merging several schemas into one integrated schema. More precisely, schema integration has been defined as "the activity of integrating the schemas of existing or proposed databases into a global, unified schema" [2]. With the current research into heterogenous databases, this process plays an important role in integrating export schemas into a global schema. [8] proposes an Entity-Relationship (ER) based federated database system where local schemas modeled in the relational, network or hierarchical models are first translated into the corresponding ER export schemas before they are integrated. In the integration of ER export schemas into a global schema, the following conflicts need to be resolved:

1. Naming conflict - Synonyms and homonyms are the two sources of naming conflicts. Renaming is a frequently chosen solution in traditional methodologies.
2. Type conflict - Same real world concept may be represented in two schemas using different modeling constructs. For example, the concept of Publisher may be modeled as an entity type in one schema and as an attribute in another schema.
3. Key conflict - Different keys may be assigned as the identifier of the same concept in different schemas. For example, attributes Ssno and Empno may be identifiers for the entity type Employee in two schemas. Given a precise known

correlation (1:1) between the two keys, this conflict is solved by asking the integrator which key to be used as the identifier in the integrated schema.

4. Constraint conflict - Two schemas may represent different constraints on the same concept. For example, an attribute Phoneno may be single-valued in one schema and multivalued in another schema. Another example involves different constraint on a relationship set such as Teach; one schema may represent it as 1:n (a course has one instructor) whereas the other schema may represent it as m:n (some courses may have more than one instructor).

Previous research has concentrated mostly on the resolution of type conflicts [1, 5, 6, 12]. Little attention has been paid to constraint conflicts. [13] identifies the roles of integrity constraints in database interoperation while [11] examines the integrity constraints that can be defined in an integrated schema. The global integrity constraints obtained can be used to optimise queries at the integrated schema level. We can reduce the average response time for global query processing by eliminating subqueries which yield empty results and formulating the global query into its optimised equivalent. Another possible use of global integrity constraints is in the validation of update transactions, preventing the formulation of subtransactions which will be rejected by the local transaction manager.

Two or more databases modeling the same real world situation, using the same data model and using the same data semantics may possess very different sets of integrity constraints based on the knowledge acquisition skills of their respective designers. We may even have conflicting constraints. This paper investigates how we can resolve the various constraint conflicts that occurs when we integrate ER schemas.

We have the following constraints in the ER model:

1. Domain (value set) constraints on the possible values that an attribute can take.
2. Attribute constraints, which specify whether an attribute of an entity type or relationship set is single-valued or multivalued.
3. Relationship constraint, which specify constraints on the participation of entity types in relationship sets.

Our approach to the resolution of these constraint conflicts is guided by the following principles:

1. Enforce the most precise constraints in the integrated schema.
2. Retrieve all the data in the local databases via the integrated schema.

Two entity types from two different schemas are *semantically equivalent* if they model the same real world concept. Real world concepts may be involved in a variety of associations called relationship sets. Two relationship sets from two different schemas are semantically equivalent if they model the same set of relationships involving the same real world concepts. The sets of instances of a pair of semantically equivalent object types (entity types or relationship sets) can be related in one of the following ways: *EQUAL, SUBSET, OVERLAP, DISJOINT.* For example, if the entity types Book from two schemas S1 and S2 model exactly the same set of books in the real world, then we have S1.Book EQUAL S2.Book. If S1 models Chinese books while S2 models English books, then we have S1.Book DISJOINT S2.Book. If S1 models all Chinese books while S2 models all Chinese and English books, then we have S1.Book SUBSET S2.Book. If S1 models all

Chinese and English books while S2 models all English and Japanese books, then we have S1.Book OVERLAP S2.Book.

The rest of the paper is organized as follows. Section 2 briefly describes the ER model. Sections 3, 4 and 5 describe the resolution of conflicts in domain constraints, attribute constraints and relationship constraints respectively.

2. The Entity-Relationship Approach

The ER approach introduced by Chen [4] attracted considerable attention in systems modeling and database design [3, 4]. The ER concepts correspond to structures naturally occuring in information systems which enhance the ability of designers to describe accurately a universe of discourse. The integration of databases in a federated database system is best performed at the conceptual model level using the ER approach [2, 10] because it has the semantics for defining all the desirable mappings.

The ER model incorporates the concepts of *entity type* and *relationship set.* An entity type or relationship set has *attributes* which represent its structural properties. An attribute can be *single-valued, multivalued* or *composite.* A minimal set of attributes of an entity type E which uniquely identifies E is called a *key* of E. An entity type may have more than one key and we designate one of them as the *identifier* of the entity type. A minimal set of identifiers of some entity types participating in a relationship set R which uniquely identifies R is called a *key* of R. A relationship set may have more than one key and we designate one of them as the *identifier* of the relationship set. If the existence of an entity in one entity type depends upon the existence of a specific entity in another entity type, then such a relationship set and entity type are called *existence dependent relationship set* and *weak entity type.* A special case of existence dependent relationship occurs if the entities in an entity type cannot be identified by the values of their own attributes, but has to be identified by their relationship with other entities. Such a relationship set is called *identifier dependent relationship set.* Existence dependent (EX) relationship sets and identifier dependent (ID) relationship sets are also called weak relationship sets. An entity type which is not a weak entity type is called a *regular entity type.* In the ER approach, *recursive relationship sets* and special relationship sets such as *ISA, UNION, INTERSECT* etc, are allowed. A relationship set which is not weak or special is called a *regular relationship set.* The structure of a database organized according to the ER model can be represented by a diagrammatic technique called an Entity-Relationship Diagram (ERD). The ERD has proven to be a useful database design tool. For more details, see [7].

3. Resolving Conflict in Domain Constraints

Conflicts in domain constraints are also known as *domain mismatch.* This occurs when we have conflict between the domains of equivalent attributes. For example, the value set for an attribute ExamScore may be in grades (A, B, C etc) in one database and in marks in another database.

There are two types of domain mismatch: *convertible* and *inconvertible* domain mismatch. While inconvertible domain mismatch is self-explanatory, we distinguish two types of convertible domain mismatch: *reversible* and *irreversible.* Examples of reversible domain mismatch (or scale differences) are 0° in Celsius corresponds to 32° in Fahrenheit, and 1 kilogram corresponds to 2.2 pounds. Mismatches of this type is easily resolved with a conversion function between the domains.

Attributes with irreversible domain mismatch are attributes whose domains are at various levels of explicitness. Examples include a grade of 'A' in one database being equivalent to a score in the range of 80 to 100 in another database, and a cuisine of 'Chinese' in one database versus 'Hunan' in another database. For mismatches of this type, each value in one domain, say A, is a sub-concept with respect to a value in another domain, say B. Hence each value in domain B corresponds to a set of values in domain A. The conversion between A and B is irreversible. We can convert from A to B, but not from B to A, denoted by $A \Rightarrow B$.

Example 1. Let entity types Restaurant in schemas S1 and S2 be semantically equivalent. Let r1 be an instance of S1.Restaurant and r2 an instance of S2.Restaurant such that r1 and r2 refer to the same real world restaurant. Let Cuisine be an attribute of Restaurant. We have r1.Cuisine = {Chinese} and r2.Cuisine = {Hunan, Cantonese}. Note that Hunan and Cantonese cuisines are Chinese cuisines. We have Domain(S2.Restaurant.Cuisine) $\Rightarrow$ Domain(S1.Restaurant.Cuisine) which indicates an irreversible domain mismatch. We can convert from the domain of S2.Restaurant.Cuisine to that of S1.Restaurant.Cuisine but not vice versa. We construct a domain mismatch hierarchy from the domains of the attributes Cuisine (Fig. 1). In the domain mismatch hierarchy, the domain of S2.Restaurant.Cuisine is at a lower level than that of S1.Restaurant.Cuisine. Note that no total order exists in the domain mismatch hierarchy.

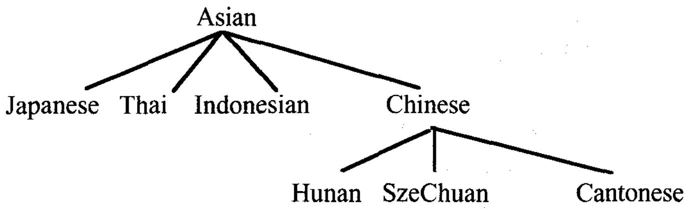

Fig. 1. Domain mismatch hierarchy for Cuisine

This irreversible domain mismatch in S1.Restaurant.Cuisine and S2.Restaurant.Cuisine can be resolved by considering the set relation between the equivalent entity types S1.Restaurant and S2.Restaurant. If S1.Restaurant EQUAL (or SUBSET) S2.Restaurant, then domain of Cuisine in the integrated schema will be that of S2.Cuisine. This ensures that we will be able to retrieve all the various cuisines via the integrated schema. On the other hand, if S2.Restaurant SUBSET S1.Restaurant, then domain of Cuisine in the integrated schema is the union of Domain(S1.Restaurant.Cuisine) and Domain(S2.Restaurant.Cuisine). This is because for all the real world restaurant instances r which are modeled in both S1.Restaurant and S2.Restaurant, r.Cuisine $\in$ Domain(S2.Restaurant.Cuisine). However, for restaurants r which are modeled in S1.Restaurant only, r.Cuisine $\in$ Domain(S1.Restaurant.Cuisine). Similarly, if S2.Restaurant OVERLAP (or DISJOINT) S1.Restaurant, then domain of Cuisine in the integrated schema is the union of Domain(S1.Restaurant.Cuisine) and Domain(S2.Restaurant.Cuisine).

The following algorithm resolves conflicts in domain constraints. If we have a reversible domain mismatch between two equivalent attributes, then it is immaterial which of the attributes' domain is used in the integrated schema. This is because a conversion function defines a one-to-one mapping between the attributes' domains.

Algorithm Resolve_DomainConstraint

```
Let $O_1$ and $O_2$ be two semantically equivalent object types in different schemas and
A be an attribute of both $O_1$ and $O_2$. $O_1$.A and $O_2$.A are semantically equivalent.
Let O be the integrated object type of $O_1$ and $O_2$ and A be the integrated attribute of
$O_1$.A and $O_2$.A.
Case 1: No domain mismatch.
    Domain(O.A) is either Domain($O_1$.A) or Domain($O_2$.A).
Case 2: Convertible domain mismatch.
    Case 2.1: Reversible domain mismatch.
        Domain(O.A) is either Domain($O_1$.A) or Domain($O_2$.A).
    Case 2.2: Irreversible domain mismatch.
        Case 2.2.1: $O_1$ EQUAL $O_2$.
            Without loss of generality, let Domain($O_1$.A) $\Rightarrow$ Domain($O_2$.A).
            Domain(O.A) is Domain($O_1$.A) to retrieve all values for attribute A via
            the integrated schema.
        Case 2.2.2: $O_1$ SUBSET $O_2$.
            If Domain($O_1$.A) $\Rightarrow$ Domain($O_2$.A)
            Then  Domain(O.A) is Domain($O_1$.A) $\cup$ Domain($O_2$.A) [1]
            Else  /* Domain($O_2$.A) $\Rightarrow$ Domain($O_1$.A) */
                  Domain(O.A) is Domain($O_2$.A) [2].
        Case 2.2.3: $O_1$ OVERLAP $O_2$ or $O_1$ DISJOINT $O_2$.
            Domain(O.A) is Domain($O_1$.A) $\cup$ Domain($O_2$.A).
Case 3: Inconvertible domain mismatch.
    Domain(O.A) is Domain($O_1$.A) $\cup$ Domain($O_2$.A).
```

After we have determined the domain of an integrated attribute, we may face the possibility of *value inconsistencies*. Consider two databases DB_1 and DB_2 held by different booksellers. Both contain an entity type Book with attributes ISBN, Title, Publisher, Price. Assuming that any domain mismatch in the attribute Price has been resolved, the same book may be priced differently by the two booksellers. Inconsistency in the attributes' values arises because ISBN $\rightarrow$ Price is a *local constraint*, which is valid in the context of a specific database only. When we

[1] For all t $\in O_1$, clearly t.A $\in$ Domain(O_1.A). On the other hand, for all t $\in O_2$ and t $\notin O_1$, t.A $\in$ Domain(O_2.A). Therefore, Domain(O.A) is Domain(O_1.A) $\cup$ Domain(O_2.A).

[2] For all t $\in O_1$ implies t $\in O_2$ since we have O_1 SUBSET O_2. For all t $\in O_2$, clearly t.A $\in$ Domain(O_2.A). Therefore, Domain(O.A) is Domain(O_2.A).

integrate the two databases, ISBN $\rightarrow$ Price is no longer true. Instead, we derive the *global constraint* {ISBN, DB} $\rightarrow$ Price where DB is a new attribute whose domain is the set of database names.

We distinguish three approaches to handle conflict in attribute values depending on the semantics of the attributes.

1. Ignore
   This indicates a situation where we do not deal with possible value conflict. We can choose any of the values. For example, the publisher of a particular book can be retrieved from either one of the databases.
2. Avoid
   Choose one of the databases as the most reliable source of values for the integrated attribute.
3. Resolve
   Case 1: Single-valued attributes
   > Value inconsistency is resolved by using a *resolution function* which derives a value(s) for the integrated attribute from the attribute values in the component databases. Examples of resolution functions include *MAX, MIN, AVERAGE, SUM* and *UNION.* For the function UNION, we may need to qualify each of the component attribute value by the database name. For example, given two booksellers' databases DB_1 and DB_2, a database integrator may choose to resolve value inconsistency in the attribute Price by keeping all the various booksellers' prices in the integrated attribute, in which case we will have the set of values {DB_1.Price, DB_2.Price} for the integrated attribute. Note that the UNION function will cause the integrated attribute to be multivalued.

   Case 2: Multivalued attributes
   > Inconsistency in the sets of values for the equivalent attributes in the component databases can be resolved by using the UNION function.

4. Resolving Conflict in Attribute Constraints

Attribute constraints are also known as attribute cardinalities. A single-valued attribute can be 1:1 (one-to-one) or m:1 (many-to-one). A multivalued attribute can be 1:m (one-to-many) or m:m (many-to-many). Attribute constraint conflict occurs when two semantically equivalent attributes do not have the same cardinalities.

Conflict in attribute constraints is resolved in two phases:

Phase 1. Establish whether the integrated attribute is single-valued or multivalued.
Phase 2. Determine precisely which type of single-valued or multivalued cardinality for the integrated attribute, that is 1:1 versus m:1 or 1:m versus m:m.

We identify six possible factors that can lead to inconsistency in attribute constraints. We first illustrate these factors informally using an example. A detailed and precise algorithm is given later in the section. Let O_1 and O_2 be two semantically equivalent object types and A be an attribute of both O_1 and O_2. Let O be the integrated object type of O_1 and O_2 and A be the integrated attribute of O_1.A and O_2.A. Suppose O_1.A is single-valued and O_2.A is multivalued.

1. Imprecise constraint design
   If for all instances t in O_2 and t.O_2.A has exactly one value, then it is possible that the multivalued cardinality of O_2.A has been imprecisely designed. We should verify the constraint design with the database integrator. If the integrator is very sure that there may exist some instance t in O_2 such that t.O_2.A has more than one value, then O.A is multivalued. Otherwise, we change the constraint of O_2.A to single-valued and the conflict is resolved.
2. Domain mismatch
   Reversible domain mismatch does not contribute to attribute constraint conflict. If we have an irreversible domain mismatch such that Domain(O_2.A) $\Rightarrow$ Domain(O_1.A), then a value in Domain(O_1.A) may correspond to a set of values in Domain(O_2.A). That is, for all $t_1 \in O_1$, $t_2 \in O_2$ such that t_1, t_2 refer to the same real world instance, all the values in t_2.A can be converted to the same single t_1.A value. No actual constraint conflict exists and O.A is multivalued. If the domains of O_1.A and O_2.A are inconvertible, then O.A is multivalued.
3. Incomplete information
   If there exist some instance t in O_2 and t.A has more than one value, then O_1.A may contain incomplete information. This occurs when O_1.A and O_2.A have exactly the same semantics. For example, both O_1.A and O_2.A may model the name of a person. However, O_2.A includes the aliases of a person. In this case, O.A will be multivalued.
4. Imprecise semantics
   If O_1.A and O_2.A do not have exactly the same semantics, then we may not have any actual constraint conflict. For example, O_1.A may model the highest qualification of a person while O_2.A may model the set of qualifications of a person. If the integrator still choose to merge these two attributes, then O.A will be multivalued. Otherwise, O_1.A and O_2.A will not be integrated.
5. Value inconsistency
   As mentioned in the previous section, value inconsistency arise because of local constraints. The integrator may choose to take the union of all the values in the equivalent attributes. In this case, O.A will be multivalued.
6. Set Relation between Object Types
   If O_1 and O_2 do not model exactly the same set of objects in the real world, then we may not have any actual constraint conflict. For example, if O_1 SUBSET O_2, then O_1.A is more restrictive than O_2.A. This may indicate that for all $t_1 \in O_1$, $t_2 \in O_2$ where t_1, t_2 refer to the same real world instance, t_2.A is single-valued. However, for some $t \in O_2$ and $t \notin O_1$, t.A is multivalued. Hence, O.A is multivalued to enable retrieval of all information in O_1 and O_2 via O. Similarly, if O_1 DISJOINT (or OVERLAP) O_2, then t.A is single-valued for all $t \in O_1$ and $t \notin O_2$, and t.A is multivalued for all $t \in O_2$ and $t \notin O_1$. Therefore, there is no actual constraint conflict and O.A is multivalued.

Some of the factors such as set relation between object types, domain mismatch and value inconsistency are orthogonal. We can have more than one factors causing a constraint conflict. The order of checking for the possible factors is important because it affects the constraint of the integrated attribute. The following algorithm determines the cause(s) of an attribute constraint conflict and resolves it.

Algorithm Check_Conflict_Cause

Let O_1 and O_2 be two object types and A be an attribute of both O_1 and O_2. Let O be the integrated object type of O_1 and O_2 and A the integrated attribute of O_1 and O_2. Let O_1.A be single-valued and O_2.A be multivalued.

```
Step 1. Check $O_2$.A for imprecise constraint design.
        If $\forall$ t $\in O_2$, t.A has exactly one value
        Then    Verify constraint design of $O_2$.A with database integrator.
                If integrator confirms imprecise constraint design
                Then    Change $O_2$.A to single-valued.
                        O.A is single-valued. Goto Step 6.
                        /* Conflict resolved. Just check for value inconsistency */
Step 2. Check for domain mismatch.
        If the domains of $O_1$.A and $O_2$.A are inconvertible
        Then    O.A is multivalued. Goto Step 6.
        Else    /* Check for irreversible domain mismatch. Reversible domain
                mismatch do not cause conflict. */
                Let K be the identifer of $O_1$ and $O_2$.
                If for each $t_1 \in O_1$, $t_2 \in O_2$, $t_1$.K = $t_2$.K and all $t_2$.A values can
                be converted to the same single $t_1$.A value
                Then    O.A is multivalued. Goto Step 5.
                        /* Check if set relation between object type is also a cause
                        of conflict. Check for value inconsistency in Step 6. */
Step 3. Check for incomplete information.
        If $O_1$.A and $O_2$.A have exactly the same semantics
        Then    Inform integrator $O_1$.A contains incomplete information.
                O.A is multivalued. Goto Step 5.
Step 4. Check for imprecise semantics.
        If $O_1$.A and $O_2$.A do not have exactly the same semantics
        Then    Inform integrator of the imprecise semantics.
                If integrator still want to integrate $O_1$.A and $O_2$.A
                Then    O.A is multivalued. Goto Step 5.
                Else    $O_1$.A and $O_2$.A will not be integrated. Exit.
Step 5. Check set relation between object types.
        If ($O_1$.A SUBSET $O_2$.A) or ($O_1$.A OVERLAP $O_2$.A) or
        ($O_1$.A DISJOINT $O_2$.A)
        Then    O.A is multivalued.
Step 6. Check for value inconsistency.
        If there exists potential value inconsistency
```

```
Then    Ask integrator for the resolution function RF.
        If RF = UNION   Then O.A is multivalued.
```

Example 2 Consider again the databases DB_1 and DB_2 held by different booksellers. Suppose we have a constraint conflict in the attribute Price: DB_1.Book.Price is single-valued and DB_2.Book.Price is multivalued. Let DB.Book.Price be the integrated attribute. In the process of determining the cause(s) of the attribute constraint conflict, we discover the following facts:

Fact 1. Cardinality of DB_2.Book.Price has been imprecisely designed because each book in DB_2 has only one selling price.

Fact 2. There is a potential value inconsistency in DB_1.Book.Price and DB_2.Book.Price because the different booksellers may price the same book differently. This is because of the local constraint ISBN $\rightarrow$ Price. The integrator removes this attribute value inconsistency by taking the union of all the prices for the integrated attribute Price.

Fact 3. DB_1.Book OVERLAP DB_2.Book.

Fact 1 automatically resolves the constraint conflict which arises because of imprecise constraint design. Therefore, we do not need to consider the OVERLAP set relation between DB_1.Book and DB_2.Book. At this point, the integrated attribute DB.Book.Price is single-valued. However, Fact 2 alerts us to a potential value inconsistency because of the local constraint. If the integrator resolves this inconsistency by taking the average selling price for the integrated attribute (Step 6 in Algorithm Check_Conflict_Cause), then DB.Book.Price remains single-valued. However, if the integrator resolves the value inconsistency by taking the union of all the prices for the integrated attribute, then DB.Book.Price becomes multivalued.

In Phase 2, we want to determine a more precise type of single-valued or multivalued attribute constraint for the integrated attribute. Given two object types O_1 and O_2 and an attribute A of both O_1 and O_2. Let O be the integrated object type of O_1 and O_2. Both O_1.A and O_2.A are either single-valued or multivalued attributes. We denote the cardinality of an attribute A by Card(A) = x:y where x, y is equal to 1 or m. Let Card(O_1.A) = 1:y and Card(O_2.A) = m:y where y = 1 or m. We derive a more precise constraint for the integrated attribute O.A as follows:

```
If $\exists$ s, t $\in O_2$ such that s $\neq$ t and s.A = t.A
Then    Card(O.A) = m:y
Else    Verify constraint design with the integrator.
        If integrator confirms imprecise constraint design
        Then    Card(O.A) = 1:y which is more precise
        Else    Card(O.A) = m:y.
```

Finally, if there is no conflict in the cardinalities of the equivalent attributes, then Card(O.A) is equal to either Card(O_1.A) or Card(O_2.A) since both cardinalities are the same. This is true except when the cardinalities of both O_1.A and O_2.A are either 1:1 or 1:m. If we have for O_1 SUBSET O_2 or O_1 OVERLAP O_2 or O_1 DISJOINT O_2, then the cardinalities of O_1.A and O_2.A are local constraints which valid in the context of their respective databases only. These constraints may not hold in the integrated database. For example, if we have O_1 DISJOINT O_2 and Card(O_1.A) = Card(O_2.A) = 1:1, then Card(O.A) = m:1 (Fig. 2).

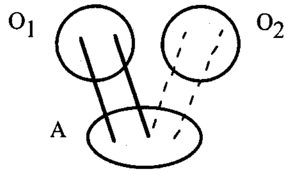

Fig. 2. The mappings from O_1 and O_2 to A are both 1:1.

We can similarly resolve any cardinality conflicts of attributes A_1 and A_2 should they belong to relationship sets R_1 and R_2 of two databases respectively. Note that our approach attempts to determine the most precise constraints in the integrated schema without compromising the retrieval of information from the local databases.

5. Resolving Conflict in Relationship Constraints

Next we proceed to resolve conflicts in relationship constraints. These are cardinality constraints on the participating entity types in a relationship set which actually indicate functional dependencies in the relationship set. Conflicts in these constraints occur when the same participating entity types of a relationship set have different cardinalities in the different databases.

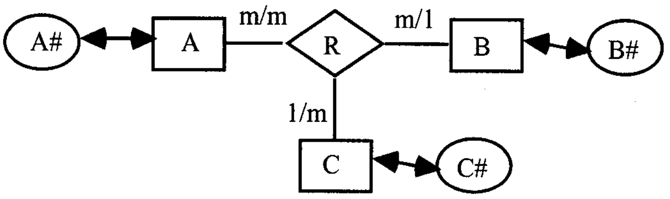

Fig. 3: A relationship set R can have more than one cardinality constraints which indicate more than one functional dependencies in R.

Fig. 3 shows a relationship set R with participating entity types A, B and C with identifiers A#, B# and C# respectively. R has two constraints as follows:

1. The first constraint where the cardinalities of A, B and C in R are m, m, 1 respectively implies that the functional dependency {A#, B#} → C# holds in R.
2. The second constraint where the cardinalities of A, B, and C in R are m, 1, m respectively implies that the functional dependency {A#, C#} → B# holds in R.

In general, each functional dependency in a relationship set represents a cardinality constraint on its participating entity types. If the identifier of a participating entity type E of a relationship set R appears on the left hand side of a functional dependency in R, then E has a cardinality of m in R with respect to that cardinality constraint in R. Otherwise, if the identifier of E appears on the right hand side of a functional dependency in R, then E has a cardinality of 1 in R with respect to that cardinality constraint in R. There is no functional dependencies in R if the cardinality of each of the participating entity types in R is m. However, the cardinality constraint of 1:1

between entity types A and B in a binary relationship set actually represents two functional dependencies A# → B# and B# → A#.

Example 3 Consider the two schemas given in Fig. 4a and Fig. 4b which models the ternary relationship between student, subject and teacher.

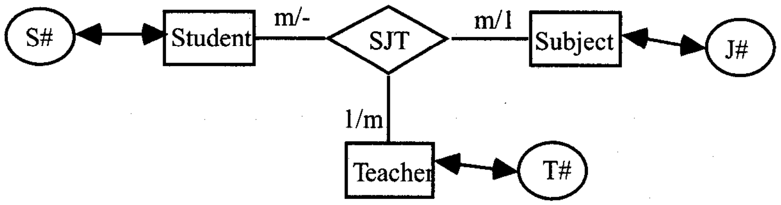

Fig. 4a: Schema S1

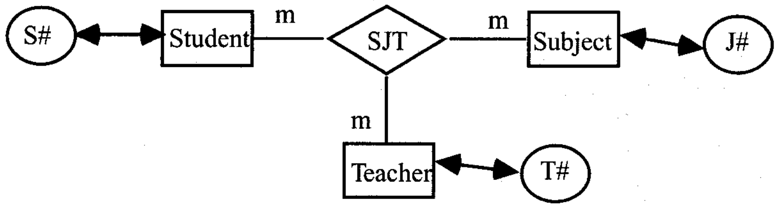

Fig. 4b: Schema S2

The following constraints apply in the relationship set S1.SJT:

1. For each subject, each student of that subject is taught by only one teacher.
2. Each teacher teaches only one subject.

From the first constraint, we have {S#, J#} → T#. From the second constraint, we have T# → J#. These functional dependencies are reflected by the two sets of cardinality constraints in S1.SJT. A dash "-" in the cardinality of the entity type Student means that it is not involved in the second constraint. We have no cardinality constraint or non-trivial functional dependency in the relationship set S2.SJT. That is, the cardinality of each of the participating entity types in S2.SJT is m.
When we integrate these two schemas, we need to reconcile these two diffferent relationship constraints. Our resolution approach will enforce the most precise constraints in the integrated schema and enable the retrieval of all the data in the local databases via the integrated schema. We examine the set relation between these two relationship sets and the functional dependencies that hold in these relationship sets. Let F_1 and F_2 be the sets of functional dependencies that hold in S1.SJT and S2.SJT respectively. $F_1 = \{\{S\#, J\#\} \rightarrow T\#, T\# \rightarrow J\#\}$ and $F_2 = \emptyset$.

Case 1: S1.SJT EQUAL S2.SJT

The integrated relationship set needs to enforce all the constraints from both S1.SJT and S2.SJT. The set of functional dependencies that hold in the integrated relationship set is $F_1 \cup F_2 = \{\{S\#, J\#\} \rightarrow T\#, T\# \rightarrow J\#\}$ which is more precise. S1 is the integrated schema. We also conclude F_1 holds in S2.

Case 2: S1.SJT OVERLAP (or DISJOINT) S2.SJT

In order to retrieve all the data in the databases modeled by S1 and S2 via the integrated schema, the integrated relationship set needs to enforce the least restricted constraints. The set of functional dependencies in the integrated relationship set is $F_1^+ \cap F_2^+$ which contains no non-trivial functional dependencies, F^+ denotes the closure of F [Maie83]. The integrated schema is S2 and there is no real constraint in the integrated relationship set. All the participating entity types in the integrated relationship set have cardinality m.

Case 3: S1.SJT SUBSET S2.SJT

A relationship in S1.SJT will need to satisfy the constraints in $F_1 \cup F_2$ while a relationship in S2.SJT but not in S1.SJT will need to satisfy the constraints in F_2 only. Therefore, in order to retrieve all the data in the databases modeled by S1 and S2, the integrated relationship set needs to enforce the constraints in F_2 only, which is the set of functional dependencies in the superset relationship set S2.SJT. The integrated schema is S2 and there is no non-trivial functional dependency in the integrated relationship set.

Case 4: S2.SJT SUBSET S1.SJT

As in Case 3, the integrated relationship set contains the same set of functional dependencies as the superset relationship set. The integrated schema is S1 and the set of functional dependencies $\{\{S\#, J\#\} \rightarrow T\#, T\# \rightarrow J\#\}$ holds in the integrated relationship set. We can also conclude that S2.SJT should have the more precise functional dependencies $\{\{S\#, J\#\} \rightarrow T\#, T\# \rightarrow J\#\}$.

From the set of functional dependencies that hold in the integrated relationship set, we can obtain the cardinality constraints of the participating entity types in the relationship set. It is easy to obtain $F_1 \cup F_2$. However, it may not be so obvious how we can obtain the cardinalites of the participating entity types from $F_1^+ \cap F_2^+$. Note that we cannot simply take the intersection of F_1 and F_2. For example, given two sets of functional dependencies $F_1 = \{A \rightarrow B, B \rightarrow C\}$ and $F_2 = \{A \rightarrow C\}$, then $F_1 \cap F_2 = \emptyset$. But $F_1^+ \cap F_2^+ = \{A \rightarrow C\}^+$.

The following proposition summarizes the resolution of relationship constraint conflicts. We assume any erroneous or imprecise constraint designs have been detected by examining the databases.

Proposition 1: Let R_1 and R_2 be two semantically equivalent relationship sets. Let F_1 and F_2 be sets of functional dependencies that hold in R_1 and R_2 respectively. Let F be the set of functional dependencies that hold in the relationship set R obtained by integrating R_1 and R_2. Each pair of semantically equivalent participating entity types from the two schemas will be merged into an entity type in the integrated schema.

Case 1: R_1 EQUAL R_2 Then $F = F_1 \cup F_2$.

Case 2: R_1 SUBSET R_2 Then $F = F_2$.

Case 3: R_1 OVERLAP R_2 or R_1 DISJOINT R_2 Then $F = F_1^+ \cap F_2^+$.

Proof: Each functional dependency in F represent a cardinality constraint among the participating entity types in the integrated relationship set.

Case 1: If R_1 EQUAL R_2 then R_1 and R_2 contain the same relationships at all points in time. A relationship r in the integrated relationship set R can be found in both R_1 and R_2. Therefore r needs to satisfy all the constraints that hold in R_1 and R_2. Hence, we have $F = F_1 \cup F_2$.

Case 2: If R_1 SUBSET R_2 then all the relationships in R_1 also exists in R_2. A relationship in R_1 will need to satisfy all the constraints in $F_1 \cup F_2$ while a relationship in R_2 but not R_1 will need to satisfy the constraints in F_2 only. Hence, we have $F = (F_1 \cup F_2) \cap F_2 = F_2$. Note that if $F_1 \subset F_2$, then clearly the set of functional dependencies in F_1 is imprecise. That is, F_2 should also hold in R_1.

Case 3: If R_1 OVERLAP R_2 or R_1 DISJOINT R_2 then a relationship r in the integrated relationship set R can be found in either R_1 or R_2. Therefore r needs to satisfy either F_1 or F_2. R will contain the least restrictive constraints which is the set of functional dependencies common in both R_1 and R_2. Hence, we have $F = F_1^+ \cap F_2^+$.

Note that unlike the resolution of attribute constraint conflicts, the resolution of relationship constraint conflicts do not require us to consider factors such as domain mismatch, incomplete information, imprecise semantics and value inconsistency. This is because these factors are either not applicable or do not influence the constraint resolution.

6. Conclusion

In this paper, we have focused on the resolution of constraint conflicts in the integration of ER schemas. We have given a detailed framework to resolve conflicts in domain constraints, attribute constraints and relationship constraints. There are two types of domain mismatch, convertible and inconvertible domain mismatch. We distinguished two types of convertible domain mismatch, namely reversible and irreversible domain mismatch. We gave an algorithm to resolve these domain constraint conflicts. We also distinguished three approaches to handle value inconsistency or conflict in attribute values depending on the semantics of the attributes.

In the resolution of attribute constraint conflicts, we identified six factors that could contribute to the conflict: imprecise constraint design, irreversible domain mismatch, incomplete information, imprecise semantics, value inconsistency and set relation between object types. We developed an algorithm to check for these various conflict causing factors and showed that the order of checking for these factors is important. In the resolution of relationship constraint conflicts, we examined the set relation between the equivalent relationship sets and the functional dependencies that hold in these relationship sets. Our conflict resolution approach does not assume that corresponding equivalent entity types or relationship sets in two schemas model exactly the same set of instances in the real world. Our approach enforces the most precise constraints and enables the retrieval all the data in the local databases via the integrated schema.

References

[1] Batini, C. and Lenzerini, M., A Methodology for Data Schema Integration in the Entity-Relationship Model, IEEE Trans.Software Engineering, SE-10, pp 650-664, 1984.

[2] Batini, C., Lenzerini, M. and Navathe, S.B., A Comparative Analysis of Methodologies for Database Schema Integration, ACM Computing Surveys, Vol 18, No 4, December 1986, pp 323-364.

[3] Chan, E.P.F. and Lochovsky, F.H., A Graphical Data Base Design Aid using the Entity-Relationship Model, in Entity-Relationship Approach to Systems Analysis and Design, North Holland, 1980, pp 295-310.

[4] Chen, P.P., The Entity-Relationship Model: Toward a Unified View of Data, ACM Transactions on Database Systems vol 1, no 1, 1976, pp 166-192.

[5] Larson, J., Navathe, S. and Elmasri, R., A Theory of Attribute Equivalence in Database with Application to Schema Integration, IEEE Trans. on Software Engineering, 15:449-463, 1989.

[6] Lee, M.L. and Ling, T.W., Resolving Structural Conflicts in the Integration of Entity Relationship Schemas, Proc. 14th Int. Conference on Object-Oriented and Entity-Relationship Modeling, 1995.

[7] Ling, T.W., "A Normal Form for Entity-Relationship Diagrams", Proc. 4th International Conference on Entity-Relationship Approach, 1985.

[8] Ling, T.W. and Lee, M.L., Issues in an Entity-Relationship Based Federated Database System, in Proceedings of the International Symposium on Cooperative Database Systems for Advanced Applications, Japan, 1996.

[9] D. Maier: Theory of Relational Databases, Computer Science Press, 1983.

[10] Navathe, S.B., Elmasri, R. and Larson, J., Integrating User Views in Database Design, IEEE Computer 19, 1, 1986, pp 50-62.

[11] Reddy, M.P., Prasad, B.E. and Gupta, A., Formulating global integrity constraints during derivation of global schema, Data & Knowledge Engineering 16, 1995.

[12] Spaccapietra, S., Parent, C., and Dupont, Y., Model independent assertions for integration of heterogenous schemas, VLDB Journal, (1), 1992, pp 81-126.

[13] Vermeer, M and Apers, P.M.G., The Role of Integrity Constraints in Database Interoperation, Proc. of the 22nd VLDB Conference, India, 1996.